Views

/*Views

A view is simply a mechanism for querying data. Unlike tables, views do not involve data storage, The marketing department, for example, may need access to email addresses in order to advertise promotions, but otherwise your company's privacy policy dictates that this data be kept secure. Therefore, instead of allowing direct access to the customer table, you define a view called customer_vw and mandate that all nonmarketing personnel use it to access customer data.*/

```
CREATE VIEW customer_vw (customer_id , first_name , last_name , email)
AS
    SELECT
        customer_id,
        first_name,
        last_name,
        CONCAT(SUBSTR(email, 1, 2),
                '*****',
                SUBSTR(email, - 4)) email
    FROM
        customer;


SELECT
    *
FROM
    customer_vw;


SELECT
    *
FROM
    customer;
```

```
describe customer_vw;
desc customer_vw;


SELECT
    first_name, COUNT(*), MIN(last_name), MAX(last_name)
FROM
    customer_vw
WHERE
    first_name LIKE 'J%'
GROUP BY first_name
HAVING COUNT(*) > 1
ORDER BY 1;


/*you can join views to other tables (or even to other views) within
a query*/


SELECT
    cv.first_name, cv.last_name, p.amount
FROM
    customer_vw cv
        INNER JOIN
    payment p USING (customer_id)
WHERE
    p.amount >= 11;
```

/*Why Use Views?

1. Data Security

You may also constrain which rows a set of users may access by adding a where clause to your view definition*/

```
CREATE VIEW active_customer_vw (customer_id , first_name , last_name , email) AS
    SELECT
        customer_id,
        first_name,
        last_name,
        CONCAT(SUBSTR(email, 1, 2),
                '*****',
                SUBSTR(email, - 4)) email
    FROM
        customer
    WHERE
        active = 1;


SELECT
    *
FROM
    active_customer_vw;
```

/*2. Data Aggregation

Reporting applications generally require aggregated data, and views are a great way to make it appear as though data is being preaggregated and stored in the database.*/

```
CREATE VIEW sales_by_film_categories AS
    SELECT
        c.name AS category, SUM(p.amount) AS total_sales
    FROM
        payment AS p
            INNER JOIN
        rental AS r ON p.rental_id = r.rental_id
            INNER JOIN
        inventory AS i ON r.inventory_id = i.inventory_id
            INNER JOIN
        film AS f ON i.film_id = f.film_id
            INNER JOIN
        film_category AS fc ON f.film_id = fc.film_id
            INNER JOIN
        category AS c ON fc.category_id = c.category_id
    GROUP BY c.name
    ORDER BY total_sales DESC;


SELECT
    *
FROM
    sales_by_film_categories;
```

/*3. Hiding Complexity

One of the most common reasons for deploying views is to shield end users from complexity.*/

```
CREATE VIEW film_stats AS
    SELECT
        f.film_id,
        f.title,
        f.description,
        f.rating,
        (SELECT
                c.name
            FROM
                category c
                    INNER JOIN
                film_category fc ON c.category_id = fc.category_id
            WHERE
                fc.film_id = f.film_id) category_name,
        (SELECT
                COUNT(*)
            FROM
                film_actor fa
            WHERE
                fa.film_id = f.film_id) num_actors,
        (SELECT
                COUNT(*)
            FROM
                inventory i
            WHERE
                i.film_id = f.film_id) inventory_cnt,
```

```
        (SELECT
                COUNT(*)
            FROM
                inventory i
                    INNER JOIN
                rental r ON i.inventory_id = r.inventory_id
            WHERE
                i.film_id = f.film_id) num_rentals
    FROM
        film f;

SELECT
    *
FROM
    film_stats;
```

/*4. Joining Partitioned Data

Some database designs break large tables into multiple pieces in order to improve performance.*/

```
/*CREATE VIEW payment_all (payment_id , customer_id , staff_id , rental_id , amount , payment_date , last_update) AS
    SELECT
        payment_id,
        customer_id,
        staff_id,
        rental_id,
        amount,
        payment_date,
        last_update
    FROM
        payment_historic
    UNION ALL SELECT
        payment_id,
        customer_id,
        staff_id,
        rental_id,
        amount,
        payment_date,
        last_update
    FROM
        payment_current;*/
```

/*5. Updatable Views

The SQL UPDATE VIEW command can be used to modify the data of a view. All views are not updatable. So, UPDATE command is not applicable to all views. An updatable view is one which allows performing a UPDATE command on itself without affecting any other table.

In the case of MySQL, a view is updatable if the following conditions are met:

• No aggregate functions are used (max(), min(), avg(), etc.).

• The view does not employ group by or having clauses.

• No subqueries exist in the select or from clause, and any subqueries in the

where clause do not refer to tables in the from clause.

• The view does not utilize union, union all, or distinct.

• The from clause includes at least one table or updatable view.

• The from clause uses only inner joins if there is more than one table or view.*/

```
UPDATE customer_vw
SET
    last_name = 'SMITH-ALLEN'
WHERE
    customer_id = 1;
```

```
/*The customer_vw view queries a single table, and only one of the
four columns is derived via an expression.*/


SELECT

    first_name, last_name, email

FROM

    customer

WHERE

    customer_id = 1;


/*While you can modify most of the columns in the view in this fashion,
you will not be able to modify the email column, since it is derived
from an expression*/


UPDATE customer_vw

SET

    email = 'MARY.SMITH-ALLEN@sakilacustomer.org'

WHERE

    customer_id = 1;


/*Views that contain derived columns cannot be used for inserting
data, even if the derived columns are not included in the statement.*/


INSERT INTO customer_vw

(customer_id,

first_name,

last_name)

VALUES (99999,'ROBERT','SIMPSON');
```

/*6. Updating COmplex Views*/

```
CREATE VIEW customer_details AS
    SELECT
        c.customer_id,
        c.store_id,
        c.first_name,
        c.last_name,
        c.address_id,
        c.active,
        c.create_date,
        a.address,
        ct.city,
        cn.country,
        a.postal_code
    FROM
        customer c
            INNER JOIN
        address a ON c.address_id = a.address_id
            INNER JOIN
        city ct ON a.city_id = ct.city_id
            INNER JOIN
        country cn ON ct.country_id = cn.country_id;

SELECT
    *
FROM
    customer_details;
```

```
/*You may use this view to update data in either the customer or
address table, as the following statements demonstrate:*/


UPDATE customer_details
SET
    last_name = 'SMITH-ALLEN',
    active = 0
WHERE
    customer_id = 1;


UPDATE customer_details
SET
    address = '999 Mockingbird Lane'
WHERE
    customer_id = 1;


/*Cannot update two tables in a single Query for Views*/


UPDATE customer_details
SET
    last_name = 'SMITH-ALLEN',
    active = 0,
    address = '999 Mockingbird Lane'
WHERE
    customer_id = 1;
```

```
/*Let's try insert*/

INSERT INTO customer_details
(customer_id, store_id, first_name, last_name,
address_id, active, create_date)
VALUES (9998, 1, 'BRIAN', 'SALAZAR', 5, 1, now());

/*Let's try insert into two tables*/

INSERT INTO customer_details
(customer_id, store_id, first_name, last_name,
address_id, active, create_date, address)
VALUES (9999, 2, 'THOMAS', 'BISHOP', 7, 1, now(),
'999 Mockingbird Lane');
```

/*Exercise - 1

Create a view definition that can be used by the following query to generate the given results:

```
SELECT title, category_name, first_name, last_name
FROM film_ctgry_actor
WHERE last_name = 'FAWCETT';*/



CREATE VIEW film_ctgry_actor AS
    SELECT
        f.title, c.name category_name, a.first_name, a.last_name
    FROM
        film AS f
            INNER JOIN
        film_category AS fc ON f.film_id = fc.film_id
            INNER JOIN
        category AS c ON fc.category_id = c.category_id
            INNER JOIN
        film_actor AS fa ON fa.film_id = f.film_id
            INNER JOIN
        actor AS a ON fa.actor_id = a.actor_id;


SELECT
    *
FROM
    film_ctgry_actor;
```

/*Exercise - 2

The film rental company manager would like to have a report that includes the name of every country, along with the total payments for all customers who live in each country. Generate a view definition that queries the country table and uses a scalar subquery to calculate a value for a column named tot_payments.*/

```
CREATE VIEW country_payments AS
    SELECT
        c.country,
        (SELECT
                SUM(p.amount)
            FROM
                city ct
                    INNER JOIN
                address a ON ct.city_id = a.city_id
                    INNER JOIN
                customer cst ON a.address_id = cst.address_id
                    INNER JOIN
                payment p ON cst.customer_id = p.customer_id
            WHERE
                ct.country_id = c.country_id) tot_payments
    FROM
        country c;

SELECT
    *
FROM
    country_payments;
```